

What Was Original About nRQL?

A Historical and Semantic Analysis

Not a claim that nRQL invented relational algebra, but an assessment of what was genuinely new in bringing projection, union, scoped negation-as-failure, and description-logic entailment together in a practical query language.

Abstract. This paper asks what, if anything, was original about the new Racer Query Language (nRQL) in the middle of the 2000s. Its answer is deliberately two-sided. Join, union, projection, subquery-like scoping, and negation-as-failure were not new primitives: relational databases, SQL, Datalog, Prolog, and F-logic already supplied close antecedents. Yet nRQL combined those ideas with description-logic entailment in an unusually explicit and practical way. It treated complex concept expressions as reasoner-backed query predicates; distinguished classical concept negation from query-level failure; allowed intermediate relations to be projected and then negated; and integrated the resulting query layer with grounded ABox matching and nonmonotonic rules. The result was an early form of epistemic relational algebra over ontology answers. The comparison with SPARQL is therefore significant but must be stated carefully. SPARQL 1.0 already had top-level projection and union, while SPARQL 1.1 later added subqueries, EXISTS/NOT EXISTS, and MINUS. Those facilities make patterns structurally close to nRQL's project-to-under neg possible in a Web standard roughly eight to nine years after the nRQL period considered here. This establishes strong technical precedence, not by itself a causal line of influence. The most defensible historical description is that nRQL was an under-credited, practical precursor to later compositional Semantic Web query languages.

Prepared from the supplied 2009 lecture slides, the nRQL/SPARQL standards chronology, and a comparative reconstruction using relational algebra, Datalog, and ontology-mediated query semantics.

August 2026

Scope, terminology, and evidence

The title is ordinary and correct English. The slightly fuller subtitle is useful because “original” can mean several different things: invention of a primitive operator, a new formal semantics, a new combination of known ideas, a practical implementation, or demonstrable influence on a later standard. This paper keeps those senses separate.

The primary source in the supplied material is Michael Wessel’s 2009 lecture deck, which presents ABox query answering, full and grounded conjunctive queries, nRQL’s `neg` and `project-to`, a multi-stage account of their evaluation, and nonmonotonic nRQL rules [1]. The deck is later than the circa-2004–2005 period at issue, but it documents the mature feature set and explicitly makes a priority claim. A contemporary nRQL paper provides the earlier historical anchor [2]. The SPARQL comparison uses the W3C Recommendations for SPARQL 1.0 and 1.1 [3, 4].

Three evidential levels are distinguished throughout:

1. **Source-supported description:** what the supplied slides visibly say or demonstrate.
2. **Semantic reconstruction:** a precise interpretation of those constructs using entailment relations, joins, projection, union, and anti-join.
3. **Historical inference:** an assessment of priority, significance, and likely reasons for limited recognition. This level is argued, not presented as a documented causal history.

Important caution. The supplied slides support the existence and intended use of the relevant operators, and they contain the authors’ own “first and only practical DL query language” claim. A universal priority claim would nevertheless require a systematic survey of every contemporary implementation and versioned release. This paper therefore uses the more defensible formulation *among the earliest practical DL query languages with compositional projection under scoped negation-as-failure*.

Contents

Scope, terminology, and evidence	i
1 The answer in one page	1
2 Historical setting: what was missing from the Semantic Web stack?	2
2.1 ABox retrieval was becoming query answering	2
2.2 The chronology in perspective	2
3 The nRQL architecture: syntax and semantics	3
3.1 A two-layer language	3
3.2 Core concrete forms	3
3.3 Grounded conjunctive queries are not full conjunctive queries	4
3.4 A relational reconstruction	5
4 The key example: projection inside negation	5
4.1 The query from the slides	5
4.2 Why not and neg must not be conflated	6
4.3 What the query does not say	6
5 How close is nRQL to other query languages?	7
5.1 Datalog with stratified negation	7

5.2	Flora-2 and F-logic	7
5.3	SQL and relational algebra	7
5.4	Epistemic DL query languages	8
6	SPARQL 1.0, SPARQL 1.1, and the project-to comparison	8
6.1	What SPARQL 1.0 already had	8
6.2	What SPARQL 1.1 added	8
6.3	Why the semantics are still not identical	9
7	What was genuinely original?	9
7.1	Not the primitives, but the composition	9
7.1.1	Reasoner-backed relations as first-class query inputs	9
7.1.2	Complex DL concept expressions in query atoms	10
7.1.3	An explicit object-level/query-level negation boundary	10
7.1.4	Projection as an internal compositional operator	10
7.1.5	Scoped NAF over projected ontology answers	10
7.1.6	Union and a genuine query algebra	10
7.1.7	Practical implementation inside a mature reasoner	10
8	Why nRQL may have received too little credit	10
8.1	Standards documents are not literature reviews	11
8.2	The communities described similar operators in different vocabularies	11
8.3	System-specific syntax can hide a general idea	11
8.4	The contribution was a package, not a single isolated theorem	11
8.5	Its grounded semantics were both practical and limiting	11
9	Priority is not the same as influence	11
10	Counterarguments and limitations	12
10.1	“SQL already had this”	12
10.2	“Datalog already had this”	12
10.3	“project-to is only column hiding”	12
10.4	“The query means all blocks are green”	12
10.5	“The first-and-only claim settles the history”	12
10.6	“Similarity proves SPARQL borrowed the idea”	12
11	Conclusion	12
A	Syntax and semantics quick reference	13
B	Worked translations	14
B.1	Provably non-green versus not provably green	14
B.2	Complex concept atom versus explicit role variable	14
B.3	Relational and SPARQL analogues of project-to	14

1 The answer in one page

Executive verdict

Yes, nRQL added something substantial. Its main originality was not the invention of projection, union, variables, or negation-as-failure considered separately. Those had long histories in databases and logic programming. The substantial contribution was the *semantic and engineering combination*:

- positive query atoms were answered by a description-logic reasoner rather than by raw table lookup;
- arbitrary supported concept expressions could occur where ordinary query languages expected an atomic unary predicate;
- query variables denoted named candidate individuals, while a concept atom could still invoke anonymous existential witnesses inside DL reasoning;
- conjunction, union, projection, and scoped negation-as-failure were composable over the resulting answer relations;
- not inside a concept and neg around a query were kept semantically distinct;
- project-to could construct an intermediate relation whose existence was then tested negatively, giving a practical counterpart of projected or correlated negated subqueries;
- the same machinery supported nonmonotonic, DL-safe rule patterns over named individuals.

The best compact description is: **nRQL was an early practical epistemic relational query language over a DL entailment service.**

Table 1: Four meanings of originality and the assessment defended here.

Question	Verdict	Reason
Did nRQL invent join, union, projection, or NAF?	no	Relational algebra, SQL, Prolog, Datalog, and F-logic supplied older counterparts.
Did it create a distinctive DL query semantics?	qualified yes	Its grounded matching and reasoner-backed atoms were distinctive; the exact semantics were hybrid and operational rather than a wholly new foundational logic.
Was the practical combination novel in DL/Semantic Web systems?	yes	The combination of complex concept atoms, explicit projection, scoped NAF, union, and reasoning-backed answers was unusually early and technically substantive.
Did it directly cause SPARQL 1.1 features?	Not established	Similarity and chronology establish precedence, but influence needs documentary evidence from design records or participants.

The historical claim is thus stronger than “nRQL happened to have a similar keyword” and weaker than “SPARQL 1.1 copied nRQL”. It is a claim about a working architecture that anticipated a later standard-language capability.

2 Historical setting: what was missing from the Semantic Web stack?

2.1 ABox retrieval was becoming query answering

Early description-logic systems were already strong at TBox classification, concept satisfiability, instance checking, realization, and simple ABox retrieval. The supplied slides explicitly frame richer ABox query answering as a recent research focus and distinguish RDF query languages such as SPARQL from “true DL query languages” such as nRQL and OWL-QL [1, slide 41]. This distinction matters: a DL query engine is expected to answer from what the ontology *entails*, not merely from what is explicitly written as triples.

For a knowledge base K , a concept expression C , a role R , and the available individual names $\text{Ind}(K)$, the elementary reasoner-backed relations are

$$\text{Cert}_K(C) = \{a \in \text{Ind}(K) \mid K \models C(a)\}, \quad (1)$$

$$\text{Cert}_K(R) = \{(a, b) \in \text{Ind}(K)^2 \mid K \models R(a, b)\}. \quad (2)$$

These relations may contain facts not explicitly asserted. They may depend on subclass axioms, inverse roles, role hierarchies, functionality, equality, transitivity, and existential reasoning.

The database question was therefore not merely “how can users write graph patterns?” It was also:

How can familiar relational operations be made to work over answer relations computed by an open-world theorem prover, while keeping object-level negation separate from failure to prove a query?

That is the design space in which nRQL should be assessed.

2.2 The chronology in perspective

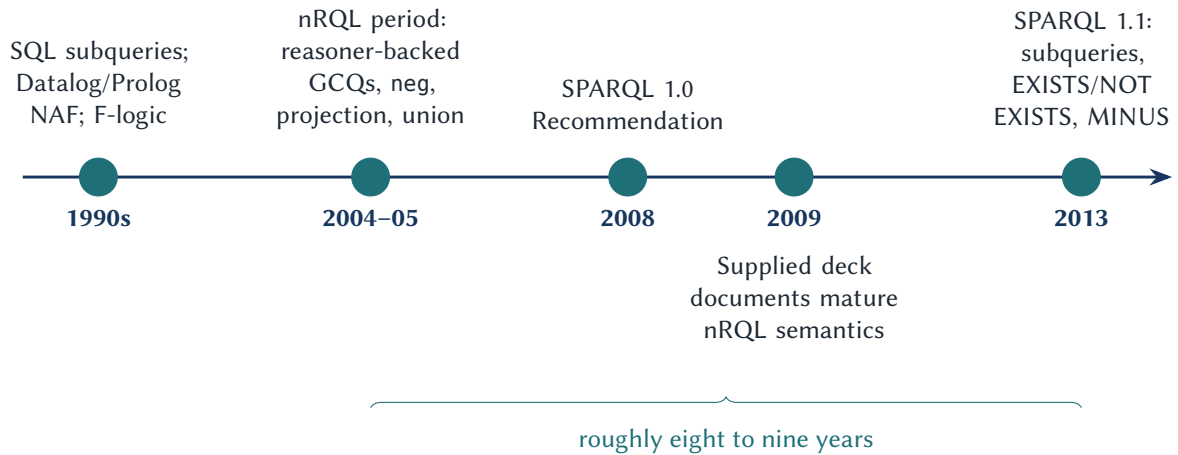


Figure 1: A conservative chronology. The exact first availability of each nRQL operator should be tied to versioned contemporary documentation; the larger sequencing is clear.

SPARQL 1.0 became a W3C Recommendation in January 2008 [3]. It already had basic graph patterns, top-level SELECT projection, UNION, OPTIONAL, and FILTER. It did not have subqueries, EXISTS/NOT EXISTS, or MINUS. Some absence tests could be simulated with the well-known OPTIONAL plus !BOUND idiom, but that was not a general, compositional replacement for locally scoped projected subqueries.

SPARQL 1.1 became a W3C Recommendation in March 2013 and added subqueries, EXISTS/NOT EXISTS, MINUS, aggregates, and other facilities [4]. From a 2004 starting point, the gap is about nine years; from a 2005 paper, about eight. That delay does not prove influence, but it makes nRQL’s earlier engineering choice historically noteworthy.

3 The nRQL architecture: syntax and semantics

3.1 A two-layer language

The most useful way to understand nRQL is as two semantic layers.

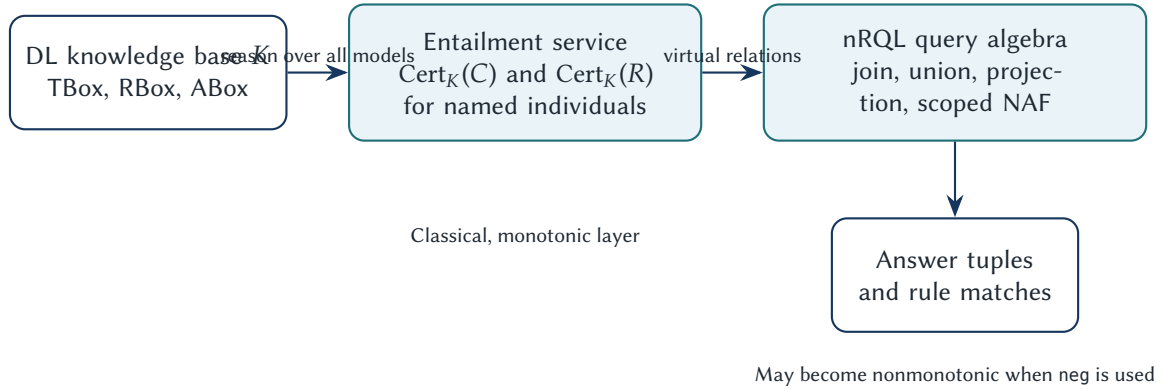


Figure 2: A semantic reconstruction of nRQL as relational operations over reasoner-computed answer relations.

The lower layer is classical DL entailment. The upper layer combines the resulting relations and, when *neg* is present, applies a closed-world test to the success or failure of a subquery. This split explains both the power of nRQL and the risk of confusing two kinds of negation.

3.2 Core concrete forms

The supplied deck gives the following core syntax [1, slides 46–57]. The union row is written schematically because the supplied slides do not display its exact concrete token; the semantic operator, not an invented spelling, is what matters here.

Table 2: Core nRQL forms and their intended readings.

Form	Relational role	Reading
(?x C)	unary reasoner atom	The binding of ?x is entailed to be an instance of concept expression C.
(?x ?y R)	binary reasoner atom	The pair of bindings is entailed to stand in role R.
(and Q1 ... Qn)	natural join	All subqueries hold for one compatible binding.
UNION(Q1, Q2)	set union	A binding is supplied by either branch, after compatible variable alignment.
(retrieve (?x ...) Q)	output projection	Evaluate Q and return the listed variables.
(project-to (?x ...) Q)	intermediate projection	Evaluate Q, retain the listed variables, and expose that projected relation to a surrounding query.
(neg Q)	scoped NAF / anti-join	Succeed for a current candidate binding when no compatible result of Q can be established.
(?x (not C))	classical DL complement	The binding is entailed to belong to $\neg C$. This is not NAF.

Variables use the familiar ?x notation, but their semantics are not simply those of first-order vari-

ables over arbitrary domain elements. In the grounded-query account emphasized by the slides, query variables are replaced by individual names and the resulting ground conjunction must be entailed. This has two consequences.

First, nRQL is naturally active-domain or named-individual oriented. Second, the concept expression in a unary atom can itself invoke full DL reasoning. For example,

```
(?x (and person
      (some has-child doctor)))
```

asks whether the reasoner entails that the named binding is a person with some doctor child. The child satisfying the existential restriction may be anonymous. Thus an anonymous witness can occur *inside* an intensional concept test even though a query variable such as $?y$ is ordinarily grounded to a name. This is one of nRQL's most characteristic blends of DL and relational semantics.

3.3 Grounded conjunctive queries are not full conjunctive queries

The supplied slides sharply distinguish a model-theoretic conjunctive query from the grounded behavior described for nRQL [1, slides 46–48]. Consider

$$q(x) \leftarrow Table(x), onTable(y, x), Block(y), Green(y), nextTo(y, z), Red(z), Block(z). \quad (3)$$

Under ordinary CQ certain-answer semantics, y and z are existentially quantified over each model. A returned name a is certain when

$$K \models \exists y \exists z \varphi(a, y, z). \quad (4)$$

The witnesses may differ from model to model and need not be named.

Under the grounded reading, a substitution σ maps every query variable to an individual name, and a match requires

$$K \models \varphi\sigma. \quad (5)$$

Omitting y and z from the top-level answer list does not automatically convert them into arbitrary model-theoretic witnesses. It merely suppresses columns of a grounded binding relation.

This difference explains the three-block example in the slides. The table is a certain answer to the full CQ because either the left or middle block witnesses the required pattern depending on the model. Yet no single grounding of all variables is entailed in every model. The grounded nRQL query can therefore return no answer even though the full existential CQ is true.

Tree-shaped queries can sometimes be *rolled up* into a complex concept expression, such as

$$Table \sqcap \exists onTable^-. (Block \sqcap Green \sqcap \exists nextTo. (Red \sqcap Block)). \quad (6)$$

Then ordinary instance retrieval recovers anonymous existential witnesses through DL semantics. The slides correctly note that rolling-up does not work generally for cycles or required coreference, and that nRQL did not automatically perform the transformation [1, slide 48].

3.4 A relational reconstruction

Let a query result be a finite relation of substitutions over named individuals. A useful semantic reconstruction is:

$$\text{Ans}_K(C(?x)) = \{\{?x \mapsto a\} \mid a \in \text{Cert}_K(C)\}, \quad (7)$$

$$\text{Ans}_K(R(?x, ?y)) = \{\{?x \mapsto a, ?y \mapsto b\} \mid (a, b) \in \text{Cert}_K(R)\}, \quad (8)$$

$$\text{Ans}_K(\text{and}(Q_1, Q_2)) = \text{Ans}_K(Q_1) \bowtie \text{Ans}_K(Q_2), \quad (9)$$

$$\text{Ans}_K(\text{union}(Q_1, Q_2)) = \text{Ans}_K(Q_1) \cup \text{Ans}_K(Q_2), \quad (10)$$

$$\text{Ans}_K(\text{project}_X(Q)) = \pi_X(\text{Ans}_K(Q)). \quad (11)$$

Negation-as-failure is best understood relative to an incoming candidate relation E . If Q shares variables with E , then a correlated anti-join is

$$\text{anti}(E, Q) = E - \pi_{\text{vars}(E)}(E \bowtie \text{Ans}_K(Q)). \quad (12)$$

A candidate survives exactly when it has no compatible result for the negated subquery. This reconstruction explains the operational behavior of the examples without pretending that `neg` is classical first-order negation.

Equation (12) is an explanatory relational model, not a claim that every RacerPro implementation detail or variable-safety rule is exhausted by one equation. Version-specific treatment of unbound variables, equality classes, duplicate elimination, and query schemas should be taken from the relevant nRQL manual.

4 The key example: projection inside negation

4.1 The query from the slides

The most historically interesting example asks for tables for which no known on-table block can be found whose greenness is not provable. In the slide syntax:

```
(retrieve (?x)
  (and (?x table)
    (neg
      (project-to (?x)
        (and (?y ?x on-table)
          (neg (?y green)))))))
```

The example contains two tables. On t_1 , blocks a and b are known green. On t_2 , c is known green, d is known non-green, and e has unknown color. The known on-table relation is

$$O = \{(a, t_1), (b, t_1), (c, t_2), (d, t_2), (e, t_2)\}. \quad (13)$$

The known-green relation contains a, b, c . Consequently, the inner NAF test retains both d and e : one is provably non-green, while the other is merely not provably green.

The crucial observation is that `project-to` is not merely display formatting. It changes a binary counterexample relation into a unary relation of tables with at least one counterexample binding. The outer `neg` then computes the complement of that unary relation relative to the current table candidates.

In nonrecursive stratified Datalog, the same structure is:

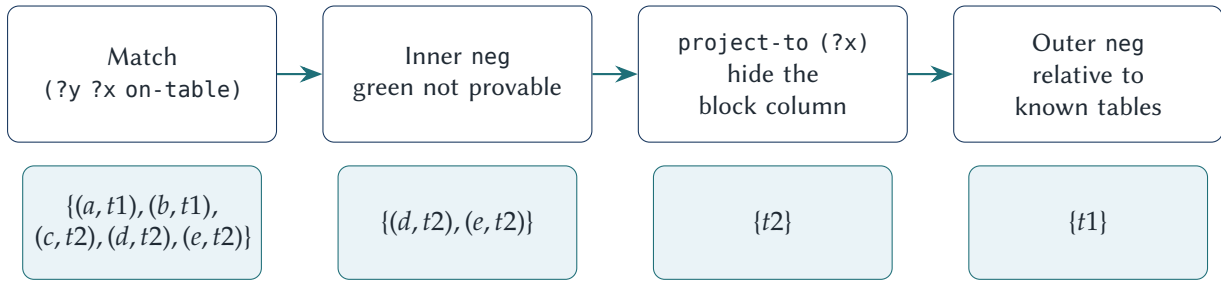


Figure 3: The four-stage semantics illustrated across slides 53–57. Projection is semantically active because it creates the unary “bad table” relation negated by the outer query.

```
bad_table(X) :-
  on_table(Y,X),
  not green(Y).

answer(X) :-
  table(X),
  not bad_table(X).
```

The structural correspondence is very close. The semantic difference is that nRQL’s positive atoms are supplied by DL entailment rather than only by a least Herbrand model of facts and rules.

4.2 Why not and neg must not be conflated

The deck explicitly contrasts blocks that can be proved not green with blocks for which green cannot be proved [1, slide 51]. In a consistent knowledge base, the distinction can be summarized as follows.

Table 3: Classical concept negation and query-level NAF.

Knowledge about a	($?x$ green)	($?x$ (not green))	(neg ($?x$ green))
$K \models \text{Green}(a)$	succeeds	fails	fails
$K \models \neg \text{Green}(a)$	fails	succeeds	succeeds
Neither is entailed	fails	fails	succeeds

The logical readings are

$$\text{not inside } C \longrightarrow K \models \neg C(a), \quad (14)$$

$$\text{neg around a query} \longrightarrow K \not\models C(a) \text{ for the current candidate test.} \quad (15)$$

The latter includes the open-world unknown case. This is analogous to the classical-negation/default-negation distinction in nonmonotonic logic programming [9], but here the two operators live in visibly different layers: one inside the DL concept language, one in the surrounding query algebra.

4.3 What the query does not say

The nested nRQL query is not equivalent to the classical DL universal restriction

$$\forall y (onTable(y, x) \rightarrow Green(y)), \quad (16)$$

or, in DL syntax, something like $\forall onTable^-.Green$. The DL universal quantifies over every domain element in every model, including anonymous individuals not represented by query names. The nRQL pattern tests the absence of a *known named counterexample binding* in the query result relation.

A useful paraphrase is therefore:

Return each known table x for which the query engine cannot establish a named y that is known to be on x and for which green cannot be established.

This is an active-domain or epistemic universal: all currently relevant named fillers pass the proof test. It is narrower than an object-level universal over the entire DL domain.

The difference is also why the query is nonmonotonic. If a new assertion $onTable(f, t_1)$ is added without a proof of $Green(f)$, t_1 can disappear from the answer. Classical positive DL entailment is monotonic; NAF over projected answers is not.

5 How close is nRQL to other query languages?

5.1 Datalog with stratified negation

At the level of query algebra, the closest conventional formalism is safe, nonrecursive, stratified Datalog with negation-as-failure. Datalog supplies:

- conjunction through rule bodies;
- projection through variables omitted from a rule head;
- union through multiple rules with the same head;
- scoped negation through lower-stratum predicates;
- active-domain variable bindings over constants.

Yet ordinary Datalog is not an exact semantics for nRQL. A Datalog atom is normally evaluated in a least or well-founded model of a rule program. An nRQL atom can instead ask an expressive DL reasoner whether a concept or role assertion follows in all models of an ontology. The closest Datalog characterization is therefore:

stratified Datalog whose extensional and intensional base predicates are oracle calls to a DL entailment service.

This oracle can sometimes be compiled away for Datalog-rewritable ontology fragments. For more expressive DLs, it is semantically essential.

5.2 Flora-2 and F-logic

Flora-2/XSB is a strong implementation-level comparison. It offers variables, rule composition, tabled evaluation, object-oriented or frame-like syntax, classical and default forms of negation, and explicit quantification. Its logical lineage is F-logic [8]. A Flora-2 program with external predicates that call Racer could reproduce much of nRQL's operational behavior.

The difference is the default semantic center of gravity. Flora-2 is a programmable rule environment whose truth values are determined by the rule program and its nonmonotonic semantics. nRQL starts from open-world DL entailment and adds relational/NAF operations around the answers. Hence the most accurate implementation analogy is *Flora-2 or XSB plus tabled DL-oracle predicates*, not Flora-2 alone.

5.3 SQL and relational algebra

SQL had nested subqueries and NOT EXISTS long before nRQL. Relational algebra had projection, union, join, and difference even earlier. These are decisive counterexamples to any claim that project - to was an unprecedented query operator in computer science.

But SQL normally evaluates one database instance under a closed-world reading. nRQL evaluates positive atoms against what follows from every model of a DL knowledge base, then applies NAF to

the resulting certain-answer relation. The novelty lies in crossing that semantic boundary in a practical system.

5.4 Epistemic DL query languages

A model-theoretically close description is provided by epistemic query languages for DLs, including later work such as EQL-Lite [10]. One can read a positive nRQL atom as an epistemic test

$$[C(a)] \text{ meaning } K \models C(a), \quad (17)$$

and query-level neg as the failure of such a knowledge test. On that reading, the table query resembles

$$[Table(x)] \wedge \neg \exists y ([onTable(y, x)] \wedge \neg [Green(y)]), \quad (18)$$

where the external quantification ranges over named query candidates, not arbitrary anonymous domain elements.

This vocabulary captures nRQL’s core idea better than “just Datalog”: it is a relational language over *known* or entailed relations.

6 SPARQL 1.0, SPARQL 1.1, and the project-to comparison

6.1 What SPARQL 1.0 already had

A fair comparison must not treat SPARQL 1.0 as a primitive triple-pattern language. It already had:

- basic graph patterns and joins;
- UNION;
- top-level SELECT projection;
- OPTIONAL and FILTER;
- a mapping-based algebra over RDF terms.

Some negative queries could be encoded by placing a pattern in OPTIONAL and filtering for an unbound variable. This idiom was useful but fragile: it depended on a suitable unbound variable, interacted with error propagation, and did not offer a general local subquery interface.

Thus nRQL was not years ahead simply because it could perform top-level projection or union. The historically stronger comparison is the combination of *intermediate projection*, *local variable scope*, and *negation of the projected relation*.

6.2 What SPARQL 1.1 added

SPARQL 1.1 added three facilities especially relevant here:

1. **subqueries**, including local SELECT projection;
2. **EXISTS** and **NOT EXISTS**, which evaluate graph patterns relative to current solution mappings;
3. **MINUS**, a mapping-difference operator with its own compatibility conditions.

The nRQL table query can be rendered in SPARQL 1.1 style without exposing the local block variable:

```

SELECT ?x WHERE {
  ?x a :Table .
  FILTER NOT EXISTS {
    ?y :onTable ?x .
    FILTER NOT EXISTS { ?y a :Green . }
  }
}

```

This pattern is structurally close to nRQL’s nested neg and project-to: the inner block variable is local to the negative pattern, while the table variable is correlated with the outer solution.

A sub-SELECT is closest to project-to as an explicit algebraic operation; a NOT EXISTS group is closest to its use under neg. The exact translation depends on which nRQL scoping and variable-safety behavior is intended.

6.3 Why the semantics are still not identical

Table 4: Operator-level similarity does not imply semantic identity.

Dimension	nRQL	SPARQL 1.1
Positive atom	DL entailment of a concept or role assertion	Match of an RDF graph pattern, possibly under a declared entailment regime
Domain of bindings	Principally named ontology individuals	RDF terms in solution mappings
Complex unary test	Native DL concept expression	Usually expanded graph pattern, property path, filter, or external entailment behavior
Classical negation	DL complement via not inside a concept	No direct general counterpart in core graph matching
Failure negation	neg over query success	NOT EXISTS, MINUS, or filtering behavior
Projection	retrieve and compositional project-to	top-level SELECT and sub-SELECT
Multiplicity	Conceptually set-oriented in this analysis	Multiset/bag semantics by default
Anonymous witnesses	May occur inside DL concept reasoning	Blank nodes and entailment regimes behave differently; ordinary variables bind RDF terms

SPARQL 1.1 therefore provides something *very close at the query-algebra level*, especially under an ontology entailment regime [5]. It is not a drop-in semantic duplicate of nRQL.

7 What was genuinely original?

7.1 Not the primitives, but the composition

Systems research often contributes by combining known ideas under constraints that make the combination non-obvious, useful, and implementable. On that standard, nRQL’s contribution was substantial.

7.1.1 Reasoner-backed relations as first-class query inputs

nRQL treated entailment results as relations to be joined, projected, unioned, and negated. This was more than exposing a reasoner’s instance-checking API. It provided a query algebra over inferred ABox

structure.

7.1.2 Complex DL concept expressions in query atoms

Ordinary Datalog and SQL begin with predicate or relation names. nRQL could place a complex concept term in the unary-predicate position. The atom itself could contain conjunction, disjunction, complement, existential or universal restrictions, inverses, and supported number restrictions. This made each atom a potentially rich theorem-proving request.

7.1.3 An explicit object-level/query-level negation boundary

The syntactic distinction between not and neg was not cosmetic. It exposed a fundamental epistemic difference:

$$K \models \neg C(a) \quad \text{versus} \quad K \models C(a). \quad (19)$$

Many application errors come from conflating these statements. nRQL made the distinction directly usable.

7.1.4 Projection as an internal compositional operator

Many languages project only at the final answer head. nRQL's `project - to` could create an intermediate relation, which could then be used under negation. This is the feature most directly comparable to later subquery scoping.

7.1.5 Scoped NAF over projected ontology answers

The nested table query realizes an anti-existential pattern:

$$\text{candidate tables} - \pi_x(\text{named counterexample bindings}). \quad (20)$$

That pattern was familiar in SQL and stratified Datalog, but its integration with open-world DL entailment and explicit classical complement was an important Semantic Web design achievement.

7.1.6 Union and a genuine query algebra

Union matters because it moves the language beyond a single conjunctive pattern. Together with join, projection, and NAF, it makes nRQL closer to a relational algebra than to a collection of reasoner API calls. The supplied slides do not show the exact union syntax, so this paper does not manufacture it; the historical feature claim should be checked against the relevant versioned manual.

7.1.7 Practical implementation inside a mature reasoner

The contribution was not only a formal possibility. The slides show nRQL as an integrated RacerPro interface alongside OWL, SWRL, and SPARQL, and use it for query answering and nonmonotonic rules [1, slides 65–66, 71]. A working implementation, optimization strategy, user syntax, and explanation of corner cases can be a major contribution even when the operators have antecedents elsewhere.

Source claim

The 2009 deck states that nRQL was “the only and first practical DL QL which can process that kind of queries” and labels `project - to` as new [1, slide 52]. The conservative historical reading is not to repeat “only” unqualified, but to recognize that the authors themselves understood compositional projection under NAF as a distinctive nRQL contribution.

8 Why nRQL may have received too little credit

The apparent absence of nRQL from later SPARQL 1.1 references is disappointing from a historical-priority perspective, but several mechanisms can explain it without implying deliberate exclusion.

8.1 Standards documents are not literature reviews

A W3C Recommendation primarily specifies interoperable behavior. Its references document normative dependencies, related standards, and selected design foundations. It is not expected to catalogue every earlier research prototype. The stronger places to search for influence would be requirements documents, issue trackers, working-group minutes, mailing-list discussions, implementation reports, and participant recollections.

8.2 The communities described similar operators in different vocabularies

nRQL spoke of ABoxes, grounded conjunctive queries, concept expressions, `neg`, and `project-to`. SPARQL spoke of RDF datasets, graph patterns, solution mappings, subqueries, `NOT EXISTS`, and `MINUS`. Database researchers would see projection and anti-join; logic programmers would see stratified NAF; DL researchers would see reasoner-backed ABox retrieval. The same structure can be independently rediscovered because each community regards its own vocabulary as primary.

8.3 System-specific syntax can hide a general idea

A construct written as

```
(neg (project-to (?x) ...))
```

can be read as a Racer-specific command rather than as a general abstraction: correlated anti-existence over a projected relation of ontology answers. Once the same pattern appears in a standardized algebra with names such as `NOT EXISTS` and `sub-SELECT`, it looks more universal, even when the earlier system embodied the idea clearly.

8.4 The contribution was a package, not a single isolated theorem

Academic credit often attaches more readily to a sharply formalized construct with a memorable theorem than to an engineering package of syntax, semantics, optimization, and implementation. nRQL's novelty was distributed across its reasoner integration, query language, manuals, and examples. That can make later citation less likely.

8.5 Its grounded semantics were both practical and limiting

Full CQ answering became a major theoretical topic. Because the nRQL account in the slides emphasizes grounded bindings, critics could view it as less general than model-theoretic CQ semantics. That criticism is valid as far as it goes, but it can obscure a different contribution: compositional relational querying over entailed named answers, including scoped NAF and projection.

9 Priority is not the same as influence

A responsible historical assessment separates four claims.

1. **Chronological precedence:** nRQL had the relevant operator combination before SPARQL 1.1. This is strongly supported by the mid-2000s chronology.
2. **Technical resemblance:** projected relations under scoped negation closely resemble later subquery and `NOT EXISTS` patterns. This is clear at the algebraic level.
3. **Conceptual anticipation:** nRQL demonstrated that a Semantic Web query language could profitably combine open-world entailment with local closed-world query operations. This is a substantive interpretive claim.
4. **Direct causal influence:** SPARQL 1.1 designers adopted the feature because of nRQL. This requires documentary evidence and is not established by similarity alone.

The distinction matters because SQL, Datalog, and logic programming provide common ancestors. SPARQL 1.1's designers could have arrived at subqueries and `NOT EXISTS` from database requirements alone. Independent convergence remains plausible.

Recommended historical wording

nRQL was an early practical ontology query language that combined reasoner-backed concept and role atoms with join, union, explicit intermediate projection, and scoped negation-as-failure. Its project-to patterns anticipated, at the level of query algebra, facilities that became standard in SPARQL 1.1 through subqueries and `NOT EXISTS`. This establishes technical precedence and makes nRQL an under-recognized precursor; it does not by itself establish direct influence on the SPARQL Working Group.

10 Counterarguments and limitations

10.1 “SQL already had this”

Correct, if the claim is primitive invention. Incorrect, if used to dismiss the DL integration. SQL did not natively make arbitrary DL concept descriptions into predicates whose extensions were computed by open-world entailment. The relevant contribution is architectural composition.

10.2 “Datalog already had this”

Also correct at the algebraic level. Datalog had implicit projection, union through multiple rules, and stratified NAF. nRQL's distinctiveness lies in exposing these operations to ontology users over reasoner-derived relations and in separating DL complement from query failure.

10.3 “project-to is only column hiding”

The slide example refutes that reading. When nested under `neg`, projection creates the relation whose existence is tested. Removing it changes the free-variable interface of the subquery and therefore changes the result. It is a scope-forming semantic operator.

10.4 “The query means all blocks are green”

Only in an operational, named-answer sense. It does not prove a universal DL restriction over all domain individuals. The slides themselves phrase the result as concerning “all known blocks”, which is the correct caution.

10.5 “The first-and-only claim settles the history”

No. It is primary-source evidence of the designers' own novelty claim, not an independent survey. The safer conclusion is “among the earliest practical systems”. A definitive priority study would examine contemporary competitors, manuals, release dates, and archived demonstrations.

10.6 “Similarity proves SPARQL borrowed the idea”

No. Shared relational and logic-programming antecedents make independent convergence credible. Direct influence is a separate archival question.

11 Conclusion

nRQL deserves more historical recognition than it usually receives. It did not invent relational projection, union, subquery scoping, or negation-as-failure. Nor did it simply reproduce SQL. Its substantial contribution was to bring those operations into a practical description-logic query environment whose

positive atoms were answered by logical entailment, whose unary predicates could be full concept descriptions, and whose syntax made a clean distinction between classical complement and failure to know.

The combination of `project-to` and `neg` was especially forward-looking. It allowed a user to build an intermediate relation of named counterexamples, project away local variables, and negate the resulting relation. That is recognizably the same query-algebra pattern later served in SPARQL 1.1 by local subqueries and `NOT EXISTS`, although the surrounding semantics differ.

The strongest defensible verdict is therefore:

nRQL was not a new foundation for query logic, but it was an original and technically substantial synthesis for ontology-mediated querying, implemented years before closely comparable facilities entered the standard SPARQL language.

Calling it an “under-credited precursor” is justified. Claiming a direct lineage into SPARQL 1.1 would require further historical evidence.

A Syntax and semantics quick reference

Table 5: Quick reference for the nRQL constructs discussed in the supplied slides.

Concrete or schematic form	Formal analogue	Important caveat
<code>(?x C)</code>	$\text{Cert}_K(C)$	C may be complex and may invoke anonymous witnesses internally.
<code>(?x ?y R)</code>	$\text{Cert}_K(R)$	Results can include inferred role pairs, not only asserted edges.
<code>(and Q1 Q2)</code>	$\text{Ans}(Q_1) \bowtie \text{Ans}(Q_2)$	Shared variables impose compatibility/coreference.
<code>UNION(Q1,Q2)</code>	$\text{Ans}(Q_1) \cup \text{Ans}(Q_2)$	Concrete spelling is left schematic because it is not shown in the supplied slides.
<code>(retrieve (?x ...) Q)</code>	$\pi_X \text{Ans}(Q)$	Shapes the final answer relation.
<code>(project-to (?x ...) Q)</code>	$\pi_X \text{Ans}(Q)$	Produces an intermediate relation available to surrounding operators; not DL existential quantification.
<code>(neg Q)</code>	anti-join / failure of Q	Nonmonotonic and epistemic; not classical complement.
<code>(?x (not C))</code>	$K \models \neg C(a)$	Object-level DL negation; unknown does not satisfy it.
Full CQ hidden variable	$\exists y$ over each model	Witness may be anonymous and vary by model.
Grounded nRQL variable	substitution by a name	A fixed ground conjunction must be entailed.
Rolled-up concept	nested $\exists R.C$	Works reliably for suitable tree-shaped patterns, not arbitrary cycles/coreference.

B Worked translations

B.1 Provably non-green versus not provably green

Provably non-green

```
(retrieve (?x)
  (and (?x block)
    (?x (not green))))
```

$$\{a \mid K \models \text{Block}(a) \wedge \neg \text{Green}(a)\}.$$

Green not provable

```
(retrieve (?x)
  (and (?x block)
    (neg (?x green))))
```

$$\{a \mid K \models \text{Block}(a), K \not\models \text{Green}(a)\}.$$

B.2 Complex concept atom versus explicit role variable

A complex concept atom

```
(?x (some has-child doctor))
```

can succeed because $K \models (\exists \text{hasChild.Doctor})(a)$ even when no named child is returned. By contrast,

```
(and (?x ?y has-child)
  (?y doctor))
```

requires a query binding for $?y$ under the grounded account. This contrast is central to understanding why nRQL is neither plain Datalog nor full CQ answering.

B.3 Relational and SPARQL analogues of project-to

Conceptually,

```
(project-to (?x)
  (and (?y ?x on-table)
    (neg (?y green))))
```

constructs

$$\pi_x\{(y, x) \in \text{Cert}_K(\text{onTable}) \mid y \notin \text{Cert}_K(\text{Green})\}. \quad (21)$$

A Datalog auxiliary predicate gives the same interface; a SPARQL 1.1 subquery gives a related local projection; a surrounding `neg` or `NOT EXISTS` performs the anti-existence test.

References

- [1] Michael Wessel. *Introduction to DLs, OWL, RacerPro*. Lecture slides, 12 January 2009. Primary supplied source; especially slides 41–57, 65–66, and 71.
- [2] Michael Wessel and Ralf Möller. A High Performance Semantic Web Query Answering Engine. In *Proceedings of the 2005 International Workshop on Description Logics (DL 2005)*, CEUR Workshop Proceedings, Vol. 147, 2005.
- [3] Eric Prud'hommeaux and Andy Seaborne, editors. *SPARQL Query Language for RDF*. W3C Recommendation, 15 January 2008. <https://www.w3.org/TR/rdf-sparql-query/>
- [4] Steve Harris and Andy Seaborne, editors. *SPARQL 1.1 Query Language*. W3C Recommendation, 21 March 2013. <https://www.w3.org/TR/sparql11-query/>
- [5] Birte Glimm and Chimezie Ogbuji, editors. *SPARQL 1.1 Entailment Regimes*. W3C Recommendation, 21 March 2013. <https://www.w3.org/TR/sparql11-entailment/>
- [6] Stefano Ceri, Georg Gottlob, and Letizia Tanca. What You Always Wanted to Know About Datalog (And Never Dared to Ask). *IEEE Transactions on Knowledge and Data Engineering*, 1(1):146–166, 1989.
- [7] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
- [8] Michael Kifer, Georg Lausen, and James Wu. Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal of the ACM*, 42(4):741–843, 1995.
- [9] Michael Gelfond and Vladimir Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–385, 1991.
- [10] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. EQL-Lite: Effective First-Order Query Processing in Description Logics. In *Proceedings of IJCAI 2007*, 2007.